ONDERZOEKSRAPPORT NR 9228

Mapping Decision Tables to Expert System Shells : An Implementation in AionDS

by

J. VANTHIENEN

G. WETS

D/1992/2376/34

Mapping Decision Tables to Expert System Shells: An implementation in AionDS

J. VANTHIENEN & G. WETS
Katholieke Universiteit Leuven
Department of Applied Economic Sciences
Dekenstraat 2, B-3000 Leuven (Belgium)
Phone: (0)16-28.58.09, Fax: (0)16-28.57.99, E-mail: fdban06@blekul11

Abstract

Building and maintaining knowledge based systems is not a trivial task. Knowledge acquisition and validation & verification efforts are major bottlenecks in the construction process. Because of the ability of the decision table technique to detect incompleteness and inconsistency, it has been recommended as a tool in verification and validation of knowledge bases.

In this paper it is argued that there is no need to restrict the use of decision tables to the transformation from expert system to decision tables, in order to obtain smooth validation or execution efficiency. In our point of view it is clearly superior to model the knowledge by means of decision tables from the start, to validate and to optimize the tables and then to implement the system, e.g. using an expert system shell. In other words, the path from decision tables to knowledge based systems is as relevant as the reverse direction.

This proposed generation strategy enables the knowledge engineer to isolate the knowledge body (the decision tables) from its implementation (which might be in the form of rules, if-then-else statements) and from the specific consultation environment, and allows to concentrate on the acquisition and modelling issues. Because the application is generated the decision tables, knowledge maintenance largely means table maintenance and the application can easily be transported to other tools or platforms.

The generation process has been implemented for AionDS and has been applied to real world applications.

Keywords

Expert systems, decision tables, decision trees, knowledge acquisition, verification & validation

1. Decision tables and Knowledge Based Systems

A decision table is a tabular representation used to describe and analyze procedural decision situations, where the state of a number of conditions determines the execution of a set of actions. Not just any representation, however, but one in which all distinct situations are shown as columns in a table, such that every possible case is included in one and only one column (completeness and exclusivity).

"A decision table is a table, representing the exhaustive set of mutual exclusive conditional expressions, within a predefined problem area." [17]

The tabular representation of the decision situation is characterized by the separation between conditions and actions, on one hand, and between subjects and conditional expressions (states), on the other. Every table column (decision column) indicates which actions should (or should not) be executed for a specific combination of condition states. An example of a decision table is given below (fig. 1).

Type of book	hard cover			normal		pocket
Wholesaler	Y		N	-		-
Quantity ordered	< 5	>=5	-	< 5	>=5	-
Special discount	x	x	x	-	x	-
Free delivery	-	x	-	-	-	-

Figure 1: Example of a decision table

In this definition, the decision table concept is deliberately restricted to the so called single-hit table, where columns are mutually exclusive. Only this type of table allows easy checking for consistency and completeness.

Decision tables and knowledge based systems show some striking similarities, although both approaches put strongly different emphases. While decision tables traditionally stress the representation facilities (with the resulting additional checking capabilities for completeness, consistency and correctness), knowledge based systems are mainly dealing with knowledge formulation (modularity, flexibility) and inference (performance, user friendliness).

Moreover, a lot of expert systems built nowadays are propositional expert systems, which are proved equivalent to decision tables [1]. All present efforts, however, to link knowledge

based systems and decision tables, focus on the transformation from expert systems to decision tables, for reasons of validation, verification or execution efficiency. In [1] e.g., expert systems are transformed into decision tables, because execution of decision tables is much faster than that of the original expert system.

In this paper it is argued that there is no need to restrict the use of decision tables to the transformation from expert system to decision tables, in order to obtain smooth validation or execution efficiency. In our point of view it is clearly superior to model the knowledge by means of decision tables from the start, to validate and to optimize the tables and then to implement the system, e.g. using an expert system shell. In other words, the path from decision tables to knowledge based systems is as relevant as the reverse direction.

2. Why do decision tables offer more than verification and validation?

Their are three domains were decision tables can be used efficiently and effectively: during the knowledge aquisition process, during the verification and validation process and as a fast way of executing the expert system.

Decision tables offer more than validation and verification. Instead of building or generating decision tables only during the validation and verification process, they will also show significant advantages in the knowledge acquisition phase itself, in which all the information has to be transformed into a coherent substance (see e.g. [8]). In the knowledge acquisition stage, a series of decision tables can be built. These tables constitute a hierarchy, as each condition or action in a decision table might be elaborated in a lower condition or action subtable respectively. In this paper it is assumed that the decision tables have been built (cf. infra) and will be converted to a full knowledge based application.

When the decision tables are transformed into an expert system shell this can be done using if-then-else statements or by using rules, as indicated in section 4. It is important to notice that after this transformation process, there will always remain a mapping between the decision tables and the expert system. Changes will be made to the decision table and immediately the expert system will be adapted. In this way maintaining the knowledge base becomes easier. The generation process has been implemented for two commercial tools, AionDS and KBMS (fig. 2).

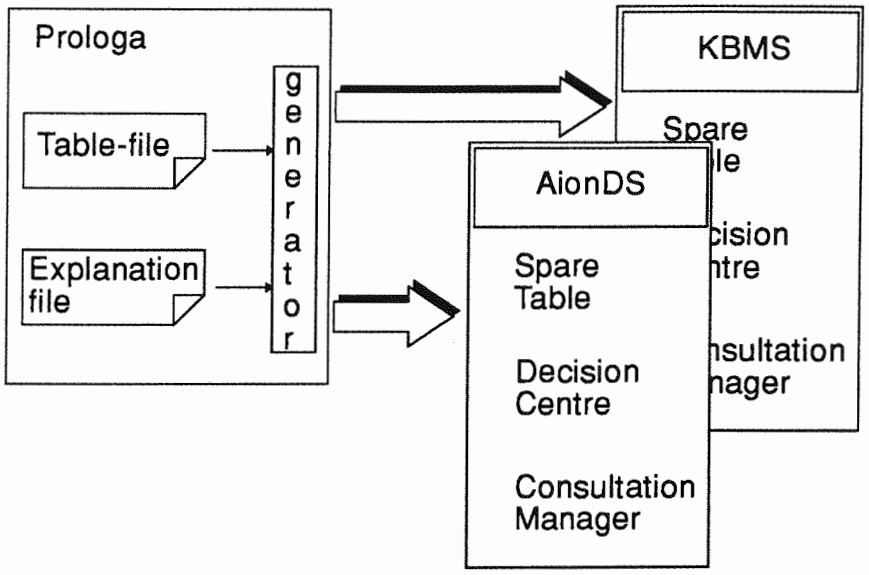


Figure 2 : AionDS and KBMS generation facility

3. Knowledge acquisition and verification & validation

3. 1. MODELLING WITH DECISION TABLES

In this phase a hierarchy of decision tables will be built modelling the domain knowledge. The decision tables might for instance be built using the PROLOGA (PROcedural LOGic Analyzer) system, an interactive rule-based design tool for computer-supported construction and manipulation of decision tables ([15], [16]). This decision table engineering workbench incorporates powerful rule based knowledge acquisition and representation, table based verification, and adequate consultation interfaces to common shells and languages. How this hierarchy of decision tables is built will not be explained here because modelling is not the main issue of this paper (see e.g. [8] for further information about modelling with decision tables.)

3. 2. VERIFICATION AND VALIDATION USING DECISION TABLES

Gathering knowledge is one of the main problems in building knowledge based systems, and usually, after the knowledge acquisition process is finished, a lot of contradictions and

insufficiencies remain to be detected and solved. Also, maintaining the knowledge base is not a trivial task which moreover, introduces unnoticed inconsistencies or contradictions.

A lot of current knowledge based systems, however, offer little or no guarantees to support validation, change and complexity control. Verification and validation of knowledge based systems are receiving increased attention [5], [6], [7], [10], [11], [13], [14].

The emerging problems of validation (to determine whether the system is designed to meet the user's needs) and verification (to determine whether the system accurately implements user specifications) have led to the occasional use of schemes, tables or similar techniques in knowledge representation and validation. It has been reported earlier (e.g. [15], [3], [12], [1]) that, in a vast majority of cases, the decision table technique is able to provide for extensive validation and verification assistance. It easily enables the designer to check for contradictions, inconsistencies, incompleteness, redundancy, etc. in the problem specification. Most of the common validation problems [9] can easily be solved using decision tables (see e.g. [16]):

- **Consistency and Correctness of Knowledge**
  Dividing knowledge over a large number of rules, designed independently, may lead to problems of inconsistency, such as: *Conflict, Cyclical rules, Invalid attribute values, Unreachable conditions.*

- **Non-redundancy of Knowledge**
  Redundancy usually does not lead to errors during consultation of the system, but it may considerably harm efficiency. The main problem with redundancy, however, is not inefficiency, but maintenance and the risk of creating inconsistencies when changing the knowledge base. Current problems are: *Subsumption, Redundant premises, Redundant rules.*

- **Completeness of Knowledge**
  No current system is able to incorporate all possible knowledge, but within the specific problem area, the following omissions often occur: *Missing knowledge, Unused attribute values or combinations, Unreachable conclusions.*

One of the major advantages of the decision table approach is that checking for completeness and consistency can already be performed during the design of the knowledge based system. At the moment validation and verification of knowledge bases takes place after the construction process. It is a clearly better solution to detect errors as soon as possible.

4. From decision tables to expert system shells

When designing or generating an application for an existing knowledge based tool, an object structure has to be developed: conditions, condition states, conclusions and table references have to be transformed to the available modelling facilities. This transition will not be considered here, as it is rather specific to the target environment, see section 5 for a transition to AionDS.

Once the object structure is built, the main problem is how to implement the decision logic knowledge which uses this object structure. The knowledge has been modelled in the form of decision tables (with proper verification and validation) and the table logic will be converted to the knowledge base. The implementation of the decision logic can be realized in two different ways. It is possible to convert the decision tables to a decision tree or to code using a nested if-then-else structure. The second way of implementing the decision tables is to convert them to a set of rules. Of course a combination of both can sometimes be useful.

When the decision logic is implemented the consultation environment is added. The consultation environment offers the necessary user and system interfaces to produce a working application. Its implementation will depend on the tool used, but is not problem-specific.

The hierarchy of decision tables is translated into a question-and-answer interface. The user, however, need not be aware of the existence of the decision tables or any relations between them. In translating the hierarchy, attention must be paid that no circular inferencing arises, that recursive table calls remain possible and that condition subtable results are properly assigned to the calling conditions.

The following demands will have to be met by the consultation environment:

- The consultation environment has to be as independent as possible from the decision tables. Therefore the environment can be built in a generic way and according to specific user interface standards.
- The user interface must allow easy communication with the user, e.g. using multiple windows, mouse support, etc.
- Prompt and help texts should be available for conditions, condition states and actions, to explain the meaning of questions and conclusions.
- At any moment a list of questions and supplied answers can be shown with the ability to change previous answers and restart the reasoning process from the specified point (WHAT IF simulations).

- At any moment it must be possible to leave the application (saving the current contents of the consultation) and restart it later. This also enables to store a list of prototype cases which can be adapted using the selective change facility.
- Maintaining the knowledge base must be easy. It should for instance be possible to plug in an updated decision table without having to generate the complete application again.

5. Concrete Transformation Prologa-AionDS

The above transformation process has been implemented in the decision table engineering workbench PROLOGA (PROcedural LOGic Analyzer, [16]). An interface was built between PROLOGA and AionDS (Aion Development System, AION), [4]. This enables the knowledge engineer to model and validate the knowledge in the form of decision tables and generate a complete consultation application in AionDS. For an interface between Prologa and KBMS (Knowledge Based Management System, AICORP), see [2].

5.1. KNOWLEDGE STRUCTURE

A. Available source elements

When transforming the decision tables, the following information is available:

- Names of conditions, condition states and actions. References to subtables are indicated in the condition or action names.
- The (contracted) decision tables on a column by column basis, where every column indicates a combination of conditions with the resulting action configuration.
- Every table can also be provided with an explanation file containing prompts, help texts or other information specific to the consultation. The explanation file is basically independent from the decision logic, such that the decision table can be altered without having to update the help information.

AionDS, being object oriented, uses the concept of classes consisting of a set of objects which have attributes called slots. An AionDS application is a hierarchy of states, where in each state rules, functions and an agenda can be defined.

B. Generation of a decision centre

Each decision table is converted to a class and condition and action variables are converted to slots, which may be linked to other conditions or actions. The decision table is converted

to if-then-else code as a function in a class. We opted for conversion to if-then-else code instead of conversion to rules for efficiency reasons. All subtables in a hierarchy are generated and linked automatically.

1. Decision Elements : Antecedents & Conclusions

Two types of data can be distinguished: condition variables and action variables. The base type of a condition variable can either be a boolean or a string. To determine a value for a condition variable two kinds of conditions must be distinguished: conditions the value of which will be determined in a subtable and conditions the value of which value will be determined by the user.

Condition determined by the user

> Slot
  ConditionName
> Base Type
  [*Boolean* | *String*]
> Facts
  [is from (*attribute constraint1*, *attribute constraint2*, ...) | is *value*]

Condition determined in a subtable

> Slot
  ConditionName
> Base Type
  [*Boolean* | *String*]
> When Sourced
  function(DTName) DTName is the name of the decision table

Three different kinds of actions can occur: a message (displaying a message), an assign (assigning a value to a slot corresponding to a condition in a subtable), a call of a decision table.

Message

> Message
  MessageName
> Message Text
  MessageName

Assign

> Slot
ActionName
> Base Type
[*Boolean* | *String*]
> When modified
Message(ActionName)

> Message
ActionName
> Message Text
ActionName is [*value1* | *value2* | ...]

Call of a decision table

Call happens directly in the decision logic with send(DTName to current).

2. Decision Logic

The knowledge of a decision table will be presented by means of if-then-else instructions which are grouped in a function. The reason why a decision table will be implemented this way rather then in a declarative way is because the reasoning process is already fixed. If rules would be used, the inference engine still has to do the reasoning process although this process was already implemented during the construction of the decision tables.

To exploit the advantages of OOP, a method will be created with the DT-function as an attached object. This method can be invoked by the instruction send(DTName to current).

> Method
DTName
> Attached Object
function(DTName)

> Function
DTName
> Function body
statements

To call the decision centre a method will be defined with as an attached object a function DTMaintableName. The main table is the decision table which is at the top of the hierarchy of decision tables.

> Method
  Dectrigger
> Attached Object
  DTMainTableName

3. Explanation files

In Prologa the length of the name of a condition or an action is very limited. This was done because the text only appears in the decision table. When decision tables are transformed to a consultation environment it may be useful to give a few words of explanation about a condition or an action. These explanations are grouped in a textfile, the XPL-file. It is important to notice that the XPL-file is completely independent of the decision logic. Changes can be made in the decision logic without changing the XPL-file.

For each condition

:Prompt
Text for the prompt of this condition in the .XPL-file. If there is no text for this condition in the .XPL-file then the default option is used: 'What is'+SlotName.

:Help
Text for the explanation of this condition in the .XPL-file.

:AionDS:
> Facts is ClassName.FunctionName

By using the :AionDS:-item it becomes possible to assign a value to a condition without asking the user for a value.

For each action

:AionDSmessage
Text of the message

:Help

Explanation of the action

:AionDS:
> When Modified
function(FunctionName)

When a value will be assigned to that action not the message ActionName will be executed, but the function FunctionName will be executed.

5.2. IMPLEMENTATION OF THE KNOWLEDGE STRUCTURE

A. General Outline

The interface offers three related generation options: a decision centre, the consultation manager and the spare table option.

Decision Centre
This option only generates the basic knowledge structure: a class with the converted actions, condition variables and decision table logic. No additional consultation facilities are provided (cf. supra). This option can be used if you want to make use of consultation options offered by the expert system shell itself or use your own specific environment.

The Consultation Manager

A full consultation environment is built together with the decision table application. An illustration of the consultation manager interface is provided in appendix. This includes the following facilities:

- *View*: a list of all variables with their values.
- *Footsteps*: a chronological survey of the questions asked, with the answer and the conclusions reached by the expert system.
- *WhatIf-mode and Reconsider*: offers the possibility in Footsteps or View to change one or more answers and restart the reasoning process.

Spare Table
This option enables the designer to plug in the decision logic of an updated table into an existing application, without having to repeat the entire generation process, and leaves the rest of the application intact.

B. The Consultation Manager

The outline of the consultation manager is as follows:

> Class
Decision Centre
  > Message for each action
  > Slot for each Condition
  > Method for each decision table
  > Attached Object: function
  > Method Showsteps
  > Function Restart
  > Function ClearListFrom
  > Function Showtable

> State
Main
> Parameter
p
> Base Type
Pointer to DecisionCentre

> Agenda
p is create(DecisionCentre)
send(DecTrigger to p->)

1. View

This option gives the user a list of all variables with their respective value. This information is immediately available so no further explanation is necessary.

2. Footsteps

This option provides the user with a chronological survey of the questions asked, with the answer and the conclusions reached by the expert system. However this information is not readily available. It is necessary to keep track of this information during the consultation. More specifically a list for each question is necessary with the following information:

variable name, Decslot in the list, the corresponding message text, DecQuestion and the value which was assigned during consultation, DecAnswer. For each conclusion reached, the action name, OutputSlot and the value of this variable, OutputValue should be saved.

This information will be added to the list when a value is assigned to that variable. For this slot a When Modified demon can be used. To this demon, a function, called an OKFunction can be attached which saves the necessary information.

Such an OKFunction looks as follows for the conditions

```
> Function
  OKConditionName
> Backward Chain
  false
> Function Body
  i is index('ConditionName', DecSlot)
  if i=0 then
  add 'ConditionName' to DecSlot
  add str(ConditionName) to DecAnswer
  else
     DecAnswer(i)=str(ConditionName)
  end
```

The OKFunction for the Actions

```
> Function
  OKActionName
> Backward Chain
  False
> Function Body
  add 'ActionName' to OutputSlot
  add str(ActionName) to OutputValue
  message(ActionName)
```

3. WhatIfmode and Reconsider

In FootSteps and View the user can select a variable to change the value of this variable. Two situations can occur:

1. The condition using this variable was not already used in the previous reasoning process. In this case there are no significant problems. When this condition becomes relevant the reasoning process will use the value of this variable.

2. When the condition using this variable was already used in the previous reasoning process the reasoning process should be reconsidered. A major problem is that the DTFunction may be running when this change occurs. To solve this problem the DTFunction should be left. This can be realized as follows.

```
> Function
  DTFunctionName
> Function Body
  If Constrained then
     Return
  End
  Statements
```

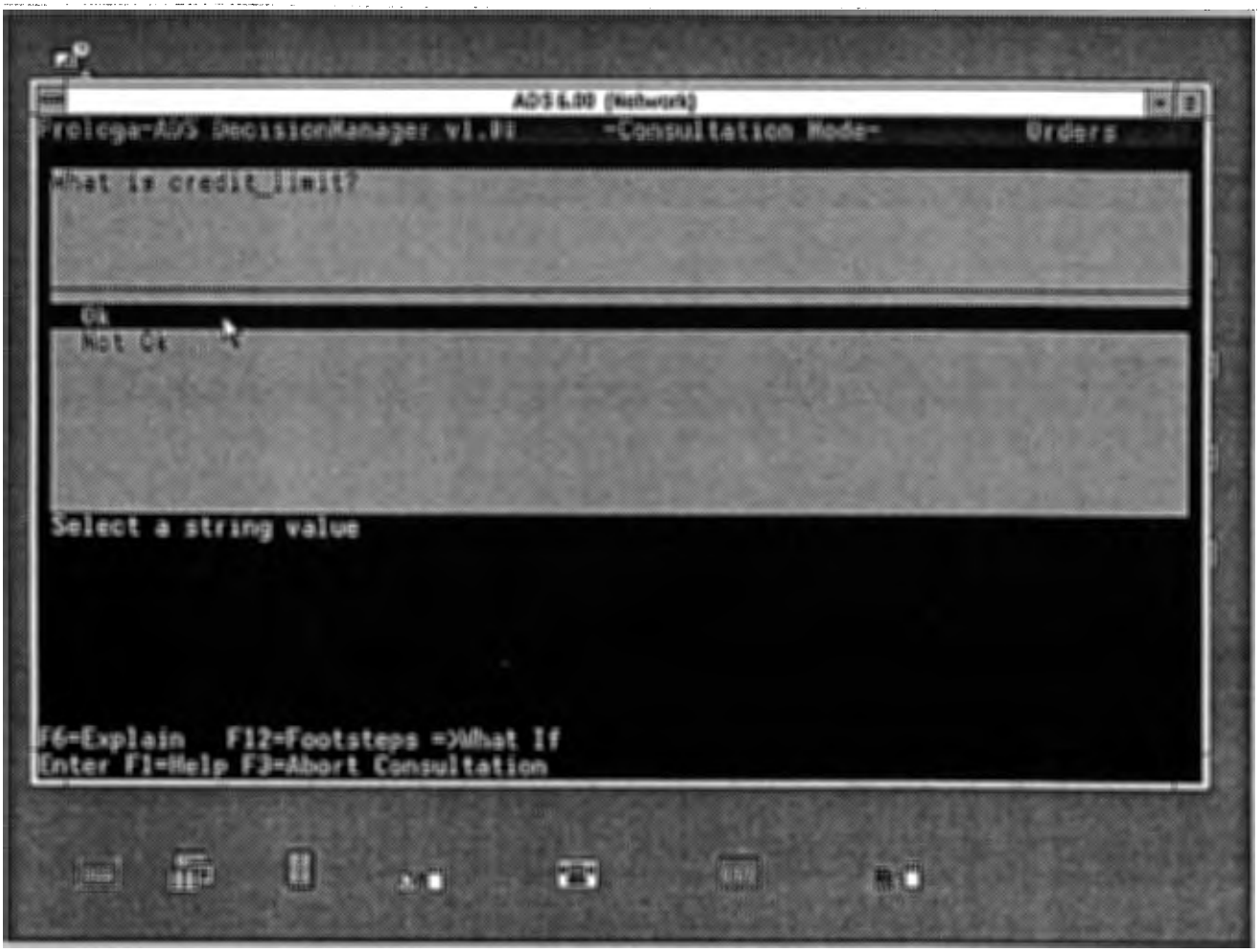
Constrained is a boolean which is set 'true' when the function should be left.

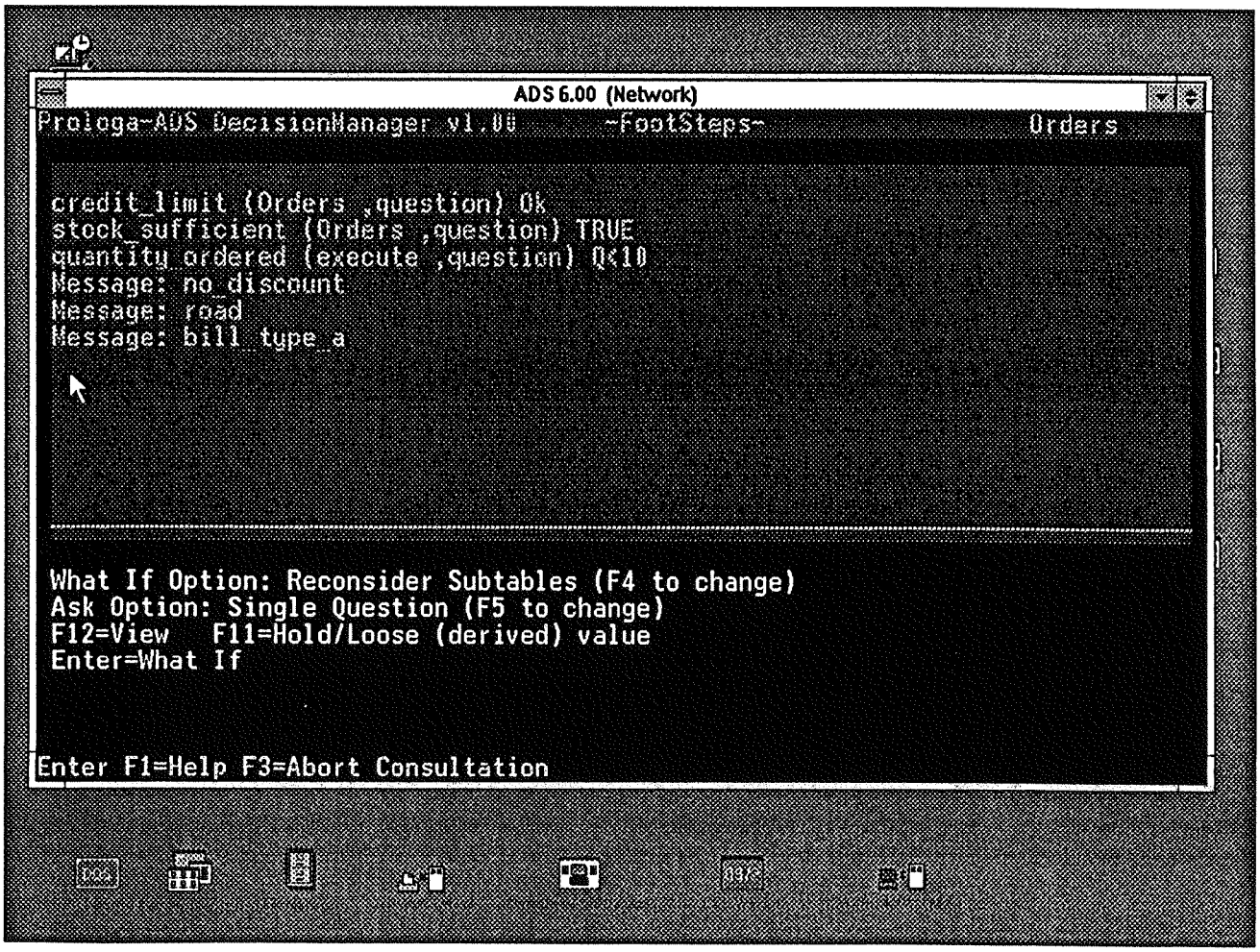
If we implement the DTFunction in this way, the function can be left, but as long as there is not a call to DTfunction the system tries to find values for those variables whose value is currently unknown. We cannot stop the system from trying to find values for certain values but when the system tries to determine a value we can initiate a When Sourced demon which calls the following function:

```
> Function
  WSConditionName
> Backward Chain
  false
> If not Constrained then
     ask(ConditionName)
  End
```

Implementing the function in this way ensures that the value of a slot will only be determined if *constrained*=false, otherwise the value for this condition will be unknown.

The use of the variable *constrained* offers also a solution to prevent the execution of an action or code. Each time a function is called by the When Sourced or When Modified demon the variable *constrained* is tested. The function will only be executed if it has the value false.


ADS 6.00 (Network)
Prologa-ADS DecisionManager v1.0i -Consultation Mode- Orders
What is credit_limit?
Ok
Not Ok
Select a string value
F6=Explain F12=Footsteps =>What If
Enter F1=Help F3=Abort Consultation


ADS 6.00 (Network)
Prologa-ADS DecisionManager v1.0i -FootSteps- Orders
credit_limit (Orders ,question) Ok
stock_sufficient (Orders ,question) TRUE
quantity_ordered (execute ,question) Q<10
Message: no_discount
Message: road
Message: bill_type_a
What If Option: Reconsider Subtables (F4 to change)
Ask Option: Single Question (F5 to change)
F12=View F11=Hold/Loose (derived) value
Enter=What If
Enter F1=Help F3=Abort Consultation

6. Conclusion

In this paper it has been shown that decision tables offer more than verification and validation. A transformation framework was developed to incorporate decision tables in expert system shells. This transformation was explained and implemented for the specific expert system shell AionDS.

Acknowledgements

The authors wish to thank M. Verhelst for his comments on an earlier version of this paper. They also greatly acknowledge the invaluable assistance of S. Jans, P. Merlevede, and G. Van Humbeeck. This research was conducted in the context of LIRIS (Leuven Institute for Research on Information Systems).

References

[1] Colomb R. and Chung C., *Very Fast Decision Table Execution of propositional Expert Systems* (Proc. AAAI90 1990) 671-676.

[2] Coucke B., *Ontwikkeling van een interface Prologa-KBMS (Knowledge Base Management System),* K.U.Leuven TEW dissertation, 1992.

[3] Cragun B. and Steudel H., *A Decision-Table Based processor for Checking Completeness and Consistency in Rule-Based Expert Systems*, Int. Journal of Man-Machine Studies 5 (1987) 633-648.

[4] Jans S., *Een interface tussen Prologa en AionDS, Toepassing van de beslissingstabellenbenadering bij het ontwerp en de implementatie van een kennissysteem,* K.U.Leuven TEW dissertation, 1992.

[5] Lalo A., *TIBRE: Un Système Expert Qui Teste Les Incoherences Dans les Bases de Règles in* (Proc. Avignon88 Vol. 3 1988) 63-84.

[6] Larsen H. and Nonfjall H., *Modeling in the Design of a KBS Validation System*, Int. Journal of Intelligent Systems 6 (1991) 759-775.

[7] Loveland D. and Valtorta M., *Detecting Ambiguity : An Example in Knowledge Evaluation*, in: Proc. Eighth Int. Joint Conf. on Artificial Intelligence, Karlsruhe, Aug. 1983, Vol. 1,182-184.

[8] Merlevede P. and Vanthienen J. , *A Structured Approach to Formalization and Validation of Knowledge*, in Proc IEEE/ACM International Conference on Developing and Managing Expert System Programs, Washington, DC, 1991, 149-158.

[9] Nguyen T., Perkins W., Laffey T. and Pecora D., *Knowledge Base Verification*, AI Magazine 2 (1987) 69-75.

[10] Nonfjall H. and Larsen H., *Detection of Potential Inconsistencies in Knowledge Bases*, Int. Journal of Intelligent Systems 7 (1992) 81-96.

[11] O'Leary T. and Goul M. et al. , *Validating Expert Systems*, IEEE Expert 3 (1990) 51-58.

[12] Puuronen S., *A Tabular Rule Checking Method*, in: Proc. Avignon87, Vol. 1, 1987 257-268.

[13] Rousset M., *Sur La Validité Des Bases de Connaissances: le Système COVADIS*, Proc. Avignon87, Vol. 1, 1987 269-282.

[14] Suwa M., Scott A. and Shortliffe E. , *Completeness and Consistency in a Rule-Based System*, in : BUCHANAN & SHORTLIFFE ,159-170.

[15] Vanthienen J., *Automatiseringsaspecten van de specificatie, constructie en manipulatie van beslissingstabellen,* K.U.Leuven Dept. Applied Econ. Doctoral Dissertation, 1986.

[16] Vanthienen J., *Knowledge Acquisition and Validation Using a Decision Table Engineering Workbench*, The World Congress on Expert Systems, Pergamon Press, Orlando (Florida), 16-19/12/91, 1861-1868.

[17] Verhelst M., *De Praktijk van Beslissingstabellen*, Kluwer, Deventer/Antwerpen, 1980.

Contents